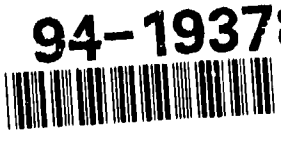
AFIT/GCS/ENG/94J-01

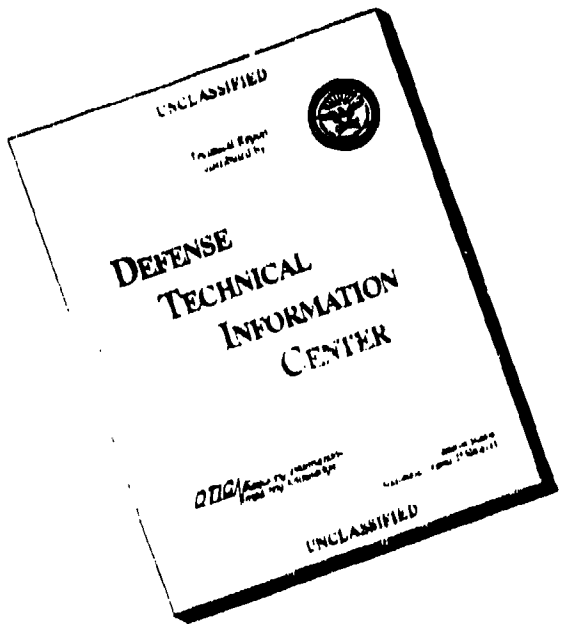
Dynamic Load Balancing
for a
Parallel Discrete-Event Battlefield Simulation

THESIS
Seth R. Guanu
Captain, USAF

AFIT/GCS/ENG/94J-01



DISCLAIMER NOTICE



THIS REPORT IS INCOMPLETE BUT IS THE BEST AVAILABLE COPY FURNISHED TO THE CENTER. THERE ARE MULTIPLE MISSING PAGES. ALL ATTEMPTS TO DATE TO OBTAIN THE MISSING PAGES HAVE BEEN UNSUCCESSFUL.

AFIT/GCS/ENG/94J-01

DYNAMIC LOAD BALANCING

FOR A

PARALLEL DISCRETE-EVENT BATTLEFIELD SIMULATION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

Seth R. Guanu, B.A.C.S.

Captain, USAF

June 14, 1994

Acknowledgements

I would like to thank my advisor, Dr. Hartrum, for his guidance and patience during this research effort. I will always remember our endless discussions in his office over various "battlefield" drawings on his blackboard. I would also like to thank my original committee members, MAJ Christensen and MAJ Sonnier for their help at the beginning of the research, and then Lt Col Hobart and Dr Potozcny for joining my committee late in the research when the other two members left AFIT.

I would also like to thank my fellow classmates, especially my BATTLESIM partner Capt Wally Trachsel. Their support and humor kept me going through my experience here at AFIT.

Most of all, I would like to thank my parents for always stressing the importance of education and hard work. Without those values, I would have never been able to succeed.

Seth R. Guanu

Table of Contents

List of Figures

List of Tables

Abstract

This thesis investigates issues involved in developing a dynamic load balancing model for a parallel discrete-event battlefield simulation. The research covers issues in task management, discrete-event simulation, parallel simulation, and load balancing.

There are four primary issues discussed concerning the design of a dynamic load balancing model. The first issue is processor load evaluation which deals with the calculation of the amount of work on a processor. The second issue is load balancing profitability determination which deals with the decision to load balance or not based on some cost-gain relationship. The third issue is task migration which deals with the selection of sources and destinations for task migration. The fourth issue deals with task selection which involves selection of appropriate tasks for efficient and effective load balancing.

As a result of the research, a dynamic load balancing model is designed that balances the work load in a parallel discrete-event battlefield simulation. The design goals used to develop this model were efficiency and maintainability of the simulation integrity. The model is then implemented and tested using AFIT's BATTLESIM program, which is a battlefield parallel discrete-event simulation.

DYNAMIC LOAD BALANCING
FOR A
PARALLEL DISCRETE-EVENT BATTLEFIELD SIMULATION

I. Introduction

1.1 Background

Battlefield simulations have become an integral part of our national defense. They provide military leaders a synthetic environment to develop doctrine, conduct training, do operational planning and rehearsal, and assess wartime situations (15:20-21). However, the size and complexity of such simulations often lead to results that take several hours or even days to compute. The lack of timely results limits the ability to examine many strategic and tactical options in a "real-time" environment. Faster methods for conducting battlefield simulations are needed to provide military commanders critical information in a timely manner. This thesis investigates dynamic load balancing as one method to achieve simulation speedup.

1.1.1 Parallel Discrete-Event Simulation. A discrete-event simulation (DES) model assumes that the system being simulated changes state instantaneously at discrete points in simulated time. The simulation model "jumps" from one state to another based upon the occurrence of an event (4:1). The three basic data structures of a DES are (4:2):

- Simulation clock - used to track how much progress has been made

- Next Event Queue(NEQ) - a time-ordered list of events which have been scheduled, but have not yet been executed
- State Variables - describe the various attributes of the system, collectively referred to as the "state" of the system being modeled.

Each event on the NEQ has a time stamp. The simulation removes the event with the smallest time stamp from the NEQ, updates the simulation clock with that time, and executes that event.

A DES can be parallelized by breaking the simulation into a set of logical processes (LPs) that communicate with each other via the sending and receiving of time stamped messages. The time stamp represents at what time the event is supposed to occur. Each LP has its own simulation clock and NEQ. (13:43-45)

1.1.1.1 PDES Difficulties. The major difficulty with PDES is the processing of events in an LP. The simulation selects the smallest time stamped event (E_{min}) from the NEQ and executes it. If it selected some other event E_X, it would be possible for E_X to modify state variables used by E_{min}. This would be simulating a system in which the future could affect the past and is unacceptable. These types of errors are known as causality errors (4:2-3). A causality error also occurs when a event is sent to an LP with a time stamp smaller than the current simulation time of that LP. PDES algorithms have been developed to address this problem.

1.1.1.2 PDES Algorithms. There is a spectrum of PDES algorithms ranging from conservative to optimistic. A conservative algorithm will strictly avoid the possibility

of a causality error occurring by determining when it is "safe" to process an event. This is accomplished by determining when all events that could affect the event in question have been processed. A optimistic algorithm will detect when a causality error has occurred and then correct it by invoking some type of rollback mechanism. (4:4) There are also algorithms that incorporate both conservative and optimistic ideas.

1.1.2 Battlefield Simulation Model. The general battlefield simulation model consists of a battlefield of a predetermined size partitioned into non-overlapping sectors. Player objects such as planes, tanks, missiles, or trucks move and interact with each other on this battlefield.

The "traditional approach" to battlefield partitioning has been static partitioning of the battlefield into uniformly sized sectors and assignment of an equal number of sectors to each LP. This is done in an effort to give each LP a somewhat equal share of the workload.

1.2 Research Problem

Many scenarios create situations during the simulation where just a few LPs are doing the vast majority of computational workload. A basic principle of military strategy is to concentrate forces in a small area. In battlefield simulations, this usually leads to a majority of the objects in one small part of the battlefield. At this point, the work among LPs becomes unbalanced.

1.3 Hypothesis

The hypothesis of this research is that dynamic readjustment of the spatial boundaries assigned to each LP should balance the work among LPs, and improve simulation speedup. The basic technique behind this concept is known as dynamic load balancing.

1.4 Scope

The specific objective of this research was to develop and test a general load balancing model for a parallel discrete-event battlefield simulation with a conservative PDES algorithm. Various load balancing techniques were analyzed. A general load balancing model was developed based on these techniques. This model was implemented and tested using BATTLESIM. BATTLESIM is a general battlefield simulation model developed at the Air Force Institute of Technology (AFIT) (1).

1.5 Assumptions

1. BATTLESIM has been tested and executes on the Intel iPSC/2 Hypercube in the AFIT Parallel Simulation Research laboratory.
2. BATTLESIM code and documentation are available.
3. BATTLESIM is representative of a general parallel battlefield simulation.
4. BATTLESIM code can easily be modified to support the load balancing strategies developed in this research.

1.6 Approach

The first step in this research was to investigate the use of dynamic load balancing in other applications and evaluate its potential in parallel battlefield simulation. This process consisted of a literature search and evaluation of previous AFIT thesis efforts.

After the literature review, the general battlefield simulation model and the underlying LP synchronization were defined. These two models served as basis for analysis of the load balancing problem.

Following the analysis stage a dynamic load balancing model was developed. This was done by evaluating the specific requirements of the general battlefield model and applying knowledge attained during the literature search.

Once the model was completed, it was implemented and tested using the BATTLESIM simulation. Various scenarios and LP configurations were used. The load balancing algorithm maintained the integrity of the simulation and showed a slight performance improvement with the benchmark scenarios tested.

1.7 Outline of Thesis

Chapter 2 of this thesis contains a literature review of current research in the area of dynamic load balancing. Chapter 3 is a analysis of the issues that were considered before arriving at a final design. Chapter 4 contains the actual design of the model and information concerning the implementation in BATTLESIM. Chapter 6 contains the results, recommendations for further research, and conclusions. Appendix A contains a description of simulation events of the general battlefield simulation model.

II. Literature Review

2.1 Introduction

Parallel processing systems have been shown to be very good at solving problems that can be broken down into tasks with uniform computation and communication patterns (18:979). There is still a large class of problems that have non-uniform computation and uneven or unpredictable communication patterns. Dynamic load balancing schemes have been researched and developed to allow parallel systems to solve these problems efficiently.

The purpose of this chapter is to provide a background in the area of dynamic load balancing as well as an introduction to BATTLESIM. This literature review discusses load balancing, a general dynamic load balancing model, various load balancing strategies, and a comparison analysis of these strategies. The information for this chapter was taken from current literature in the areas of parallel processing, load sharing, task scheduling, and task migration. The paper by Willebeek-LeMair (18) provides a good overview of similar work done in the field (5) (3) (8) (12).

2.2 Load Balancing

Load balancing focuses on maximizing the utilization of all processors in a parallel computing system. There are two types of load balancing, static and dynamic. Static load balancing includes the partitioning, allocation, and scheduling of tasks before runtime. In contrast, dynamic load balancing uses various system data to redistribute tasks within the system during runtime. (11:137-138)

2.3 General Dynamic Load Balancing Model

Willebeek-LeMair and Reeves have developed a general model for dynamic load balancing (18:980). The four phases are:

1. Processor Load Evaluation
2. Load Balancing Profitability Determination
3. Task Migration Strategy
4. Task Selection Strategy

2.3.1 Processor Load Evaluation. During this phase, the current load is estimated for each processor in the system. These load values are then sent to the load balancer to determine load imbalances and make load sharing decisions.

2.3.2 Load Balancing Profitability Determination. Once a load imbalance has been detected, it must be determined if the potential speedup gained from load balancing is greater than the overhead caused by load balancing itself.

An important calculation in this process is the load imbalance factor O(t). O(t) is defined as the difference between the maximum processor loads before and after load balancing, L_{max} and L_{bal}, respectively, at time t ($O(t) = L_{max} - L_{bal}$) (18:980).

2.3.3 Task Migration Strategy. If it is determined profitable to load balance the system, then sources and destinations for task migration are determined. The sources are informed of the quantity and destination of tasks for load balancing.

2.3.4 Task Selection Strategy. The source processors, once informed of the quantity and destination of tasks, must determine the best tasks for efficient and effective load balancing and send them to the appropriate destinations.

2.4 Dynamic Load Balancing Strategies

2.4.1 Gradient Model. The gradient model is driven by under loaded processors in the system. Under loaded processors inform other processors of their state and the overloaded processors in the system respond by sending a portion of their load to the nearest lightly loaded processor. As a result, tasks in the system tend to migrate towards the under loaded points. (18:981-982)

Two very important threshold parameters in this system are the Low-Water-Mark (LWM) and the High-Water-Mark (HWM). A processor is considered light if its load is below the LWM and heavy if it is above the HWM. (18:981-982)

The proximity of a node is defined as the shortest distance from itself to the nearest lightly loaded node in the system. Initially all nodes have a proximity of w_{max}, which is a constant equal to the diameter of the system. A node with a light load has a proximity of zero. All heavy nodes p with near-neighbors, n_i, have a proximity as:

$$proximity(p) = min(proximity(n_i)) + 1.$$

The system is saturated if all processors report a proximity of w_{max} and therefore does not need load balancing. When the proximity of a node does change, it must notify its

near-neighbors. The load balancing process is initiated when a node reports a proximity of zero (has a light load).

2.4.2 Sender Initiated Diffusion (SID). SID uses a local, near-neighbor diffusion approach which employs overlapping balancing domains to achieve global balancing (9:17-18). The scheme is purely distributed and asynchronous because each processor acts independently, sending excess work to under worked neighbors. Willebeek-LeMair and Reeves show that for an N processor system with a total system load L unevenly distributed across the system, a diffusion approach, such as the SID strategy, will eventually cause each processor's load to converge to L/N (18:982-983).

Each processor periodically receives a load update message from each neighbor in its domain. If a processor p receives a message indicating that a neighbor's load, L_{neigh} is less than L_{LOW}, where L_{LOW} is a preset threshold, and its own load L_p is higher than the average load in the domain by a certain threshold, $L_{threshold}$, then load balancing is initiated. (9, 18)

2.4.3 Receiver Initiated Diffusion (RID). The RID approach is often thought of as the converse of the SID approach because it is receiver initiated instead of sender initiated. It is similar to SID in that it implements balancing domains.

Each processor receives load update messages from neighbors in its domain. If a processor's load drops below a pre-specified threshold L_{LOW}, and is below the average domain load by more than a pre-specified amount $L_{threshold}$, then the load balancing process

is initiated. This consists of requesting work from overloaded neighbors in its domain. (9, 18)

2.4.4 Hierarchical Balancing Method (HBM). The HBM decentralizes the balancing process by organizing the system into a hierarchy of balancing domains similar to a binary-tree (9:19-20). One processor at each level is designated to control the balancing operations.

Processors in charge of balancing at a level l_i, receive load information from both lower level, $l_{(i-1)}$, domains. Global balancing for the system is achieved by ascending the tree and balancing the load between adjacent domains at each level in the hierarchy. Balancing is initiated within a domain whenever the domain's designated controller detects an imbalance. The tree structure is an attractive scheme to organize the nodes because it minimizes the communication overhead and can be adapted to accommodate large systems. (9, 18)

2.4.5 Dimension Exchange Method (DEM). The DEM is very similar to the HBM scheme in that small domains are balanced first and then combined to form larger domains until the entire system is balanced. It is different than HBM in that it is a synchronized approach.

Load balancing proceeds as follows (18):

1. All processor pairs in the first dimension balance the load between themselves.
2. All processor pairs in the second dimension balance the load between themselves.
3. All processor pairs in the N th dimension balance the load between themselves.

4. All processor pairs in the N+1 th dimension balance the load between themselves, etc.

The balancing is initiated by any processor which has a load that is less than a preset threshold, $L_{threshold}$. The synchronization approach used in DEM ensures that the entire system will achieve a balanced load.

2.4.6 Load Update Strategy. The load update strategy is very important to most dynamic load balancing schemes. The majority of dynamic load balancing strategies, including the ones discussed here, make load balancing decisions based on the load levels of various processors in the system.

Willebeek-LeMair and Reeves (18) state that although the degree of knowledge may vary from one strategy to another, the quality of information governs the intelligence of the load balancing decision. The quality of information relies on three key factors:

1. The accuracy of processor load estimates
2. The aging of information due to the communication latency of the interconnection network and the destination of load information
3. The frequency of the load update messages

The first factor is application dependent and usually involves a tradeoff between the quality of the estimate and the complexity of the estimation process. The second factor is dependent on the machine architecture and the type of load balancing scheme used. The third factor is the most flexible of the three, and is used to tune the performance of the particular load balancing strategy used.

2.5 *Comparison Analysis*

Table 1 provides a general overview of the differences between the various dynamic load balancing techniques discussed in this chapter (18:987). The selection of a dynamic load balancing technique should be based on the application and the hardware architecture.

The following points can be made about the strategies (9, 18):

- The DEM strategy tends to outperform the other schemes.
- The performance of the DEM and HBM schemes depend heavily on the system interconnection topology. The hypercube topology is ideally suited for these two schemes.
- The overhead costs of the DEM and HBM schemes degrade their performance when the number of nodes is large (1000 processors).
- The RID strategy is easily ported to simpler topologies and can be expanded gracefully for larger systems.

Table 1. Summary of Comparison Analysis

Category	GM	SID	RID	HBM	DEM
Initiation	receiver	sender	receiver	designated	designated
Balancing Domain	variable	overlapped	overlapped	variable	variable
Knowledge	global	local	local	global	global

2.6 BATTLESIM

BATTLESIM is a high-resolution parallel discrete-event battlefield simulation developed by the parallel simulation research group at AFIT. It uses a conservative PDES algorithm. It was designed to simulate objects (aircraft, missiles, tanks, and trucks) moving along predetermined route points until they sense another object or a battlefield sector boundary. When an object senses another object, it reacts by either attacking, evading, or continuing its present course. Boundary crossings are handled by boundary crossing events which involve either object replication into a sector because the object has visibility there, object removal from a sector because it no longer has visibility there, or updating object ownership because the object physically resides in a new sector. (1)

Figure 1 illustrates the BATTLESIM system overview. Multiple scenario input files are created by the user and supplied to BATTLESIM at run time. Each scenario file contains various objects, their associated attributes and a list of route points. The user also supplies a mapping file which describes the sector-to-LP mappings.

Once the simulation is executing, BATTLESIM generates object status information which is sent to a graphics file as well as the screen. The graphics file can be sent to a graphics workstation which will depict the battlefield activity.

2.7 Conclusion

This literature review provides a background into the area of dynamic load balancing and BATTLESIM. Of particular importance is the general load balancing model which is used as basis for the requirements discussion in the following chapter.

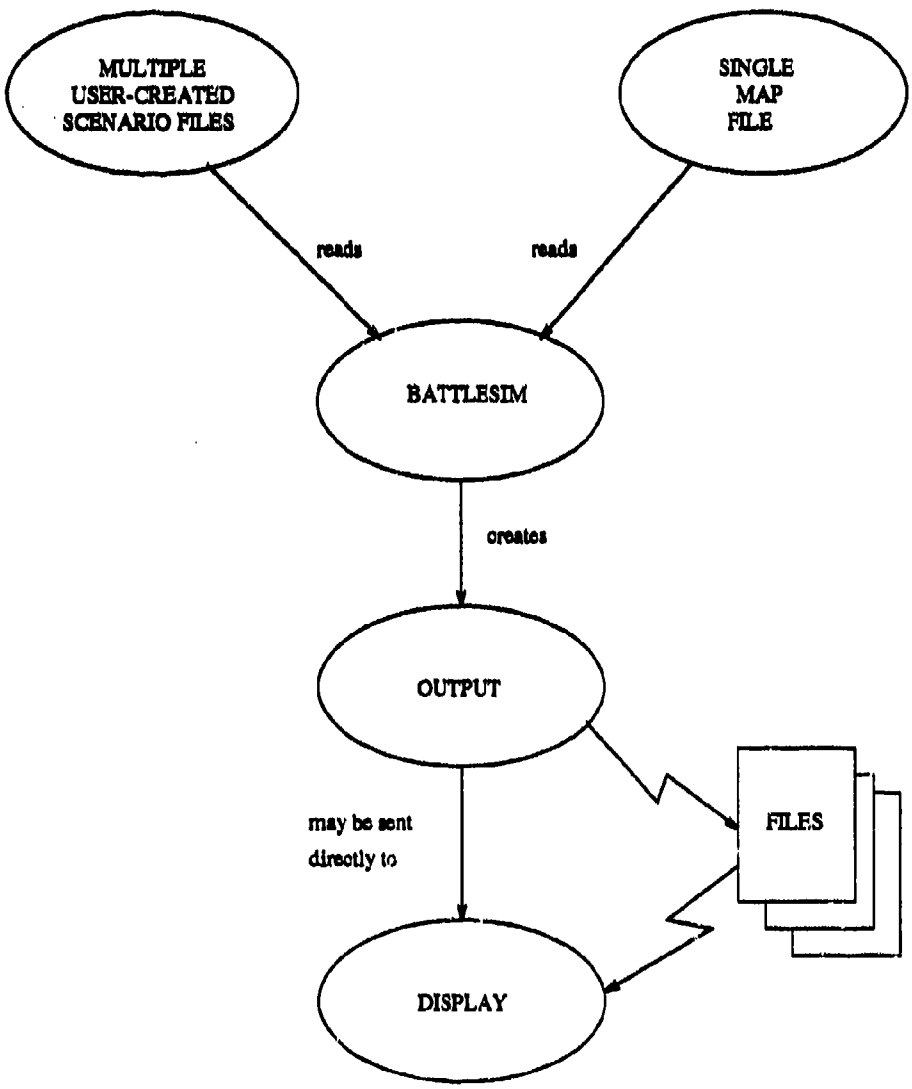


Figure 1. BATTLESIM "Big Picture"

III. Analysis of Design Issues

3.1 Introduction

The design of the dynamic load balancing model (DLBM) in this research required addressing many issues. These issues included:

- The parallel discrete-event simulation model
- The general battlefield model
- The load balancing model
- Development of benchmark scenarios

This chapter discusses the major design issues. First the parallel discrete-event simulation model and the general battlefield model used are defined. Next the design issues of the load balancing are discussed. The last section discusses the development of benchmark scenarios.

3.2 Parallel Discrete-Event Simulation Model

3.2.1 Architecture. The PDES model used in this research consists of set of communicating LPs. Each LP is a discrete-event simulation itself with its own simulation clock, next event queue, and state variables. This research uses a 1:1 mapping between the LPs and the physical nodes or processors. LPs communicate with each other via the sending and receiving of time stamped messages. Arcs, otherwise known as channels, define which LPs can communicate with each other. Input arcs to LP_i define a subset of LPs from which LP_i can receive messages. Correspondingly, output arcs from LP_i define a subset of LPs to which LP_i can send messages.

3.2.2 LP Synchronization. A conservative protocol is used to avoid deadlock and causality errors. The protocol used is a variant of the Chandy-Misra approach with NULL messages (2, 17). This protocol uses NULL messages (messages with only a time stamp) to specify the earliest time at which the next message could be sent, allowing the receiving LP to process up to that time. Each arc has a channel clock that records the time of the last message sent or received on that particular arc.

The protocol prevents causality errors by not allowing the LP to update its local simulation clock until it has received a message on each of its inputs with a time greater than or equal to its own simulation time. Once this constraint is satisfied, the LP identifies the smallest time stamp of these message and updates its local simulation clock to this time. The LP then executes all events in its next event queue with a time less than or equal to its local simulation clock. (17)

The concept of lookahead(event prediction) allows this protocol to improve its performance. For example, if an LP simulates a car wash that takes three time units to wash a car and five to wash a truck, the minimum time delay is three time units. Using the minimum time delay, if the LP time is ten and a vehicle arrives at time ten, it will not be finished any earlier than time 13. Thus the LP could send a NULL message with a time stamp of 13. (17)

There are two points at which NULL messages are sent (17):

1. At startup - NULL messages are sent to each output arc with the min delay time of that arc (different arcs can have different delay times).

2. When an LP receives a NULL message, NULL messages are sent on all output arcs with a time stamp of min(t_NEQ, $safetime + min\_time\_delay$).

3. When an LP sends a real message with a time stamp of t_time to another LP, NULL messages are sent on the remaining output arcs with a time stamp of t_time.

3.3 General Battlefield Model

The general battlefield model consists of player objects such as planes, tanks, missiles or trucks interacting on a battlefield. This battlefield is of a predetermined size and partitioned into non-overlapping sectors. The following section provides a hierarchy of the model.

3.3.1 Hierarchy.

The battlefield model is organized as follows:

- the **battlefield** is partitioned into **sectors** based on spatial dimensions
- an **LP** contains adjacent sectors
  - a sector is mapped to one and only one LP, but one LP can have multiple sectors mapped to it
- a **sector** object contains exactly one **playerset**
- a **playerset** object consists of **players**, which can be
  1. Actual players
  2. Player copies - copies of players from other playersets
- a **player** object consists of several attributes including:

- Object type
- Object identifier
- Current time
- Spatial location on battlefield
- Velocity
- Sensor range

There are three maps used for maintaining the player objects in the simulation (16):

- PLAYER_PTR_TO_SECTOR_ID - this map is used to maintain the relationship between each copy of a particular player and the sector id that it resides in.
- PLAYER_ID_TO_OWNERS_SECTOR_ID - this map is used to maintain the relationship between a player id and the sector id that the owner copy of the player resides on.
- PLAYER_ID_TO_COPIES_SECTOR_ID - this map is used to maintain the relationship between the owner copy of a player and its other copies. This is accomplished by mapping the player id of a player to each sector id that contains copies of the player.

3.3.2 Battlefield Partitioning. Sectors were originally incorporated into battlefield simulations to reduce the time is takes to predict a next event for a particular player. The absence of sector partitioning requires the simulation to search all players in the battlefield when predicting the next event for a particular player. Sectoring the battlefield allows the simulation to search only players in given sector when predicting the next event.

Player copies in the appropriate sectors allow the simulation to reduce its search space from the whole battlefield to just one sector.

Sectors can conceptually be any size or shape as long as they are no larger than the battlefield itself. Some commonly used sector shapes are hexagons, rectangular cubes, squares, and strips (see Figure 2) (1). The choice of which shape to use as a sector is often determined by the specific application. For example, Moser used strips to partition the area in which balls were travelling in his pool table simulation (14), while Bergman allowed rectangles in BATTLESIM (1).

This research considered using squares and strips as the sector shape. Analysis of the square and strip sector shapes with respect to the general battlefield simulation provided the following observations:

- **player copies** - both shapes had a best case of zero player copies. The strip could have at most one player copy per player. The square could have at most three player copies. This would occur when a player was located in the corner of the square and sensing into three other adjacent squares.
- **communication costs** - these are based on the number of LPs with which a sector would have to communicate. Both shapes had a best case of zero. This best case would occur when all sectors in the battlefield were assigned with to one LP. The worst case occurs when each sector is mapped to a different LP. When this happens, the strip has to communicate with two LPs while the square must communicate with eight.

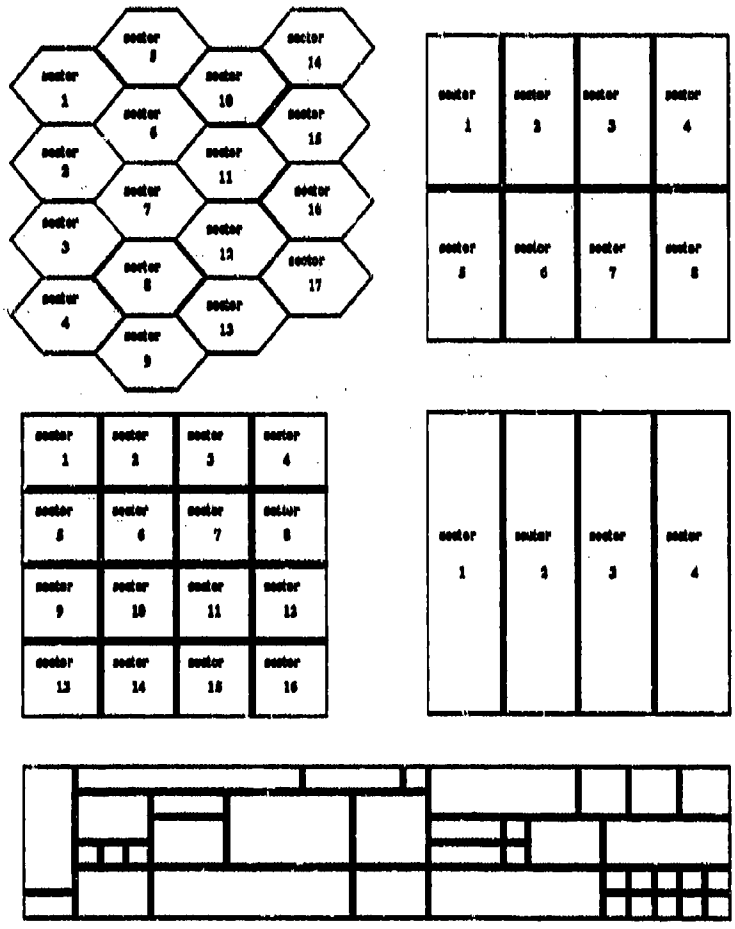


Figure 2. Examples of How to Partition a Battlefield

Strips were selected as the sector shape for this research because they potentially required fewer player copies and communication paths, while the squares did not have any distinct advantage that would compensate for those costs.

3.3.3 Execution of Model. Execution of model starts with the simulation scheduling an initial event for each player. From this point on, whenever the simulation executes an event for a player, it also determines the next event for that player and schedules it. This process continues until the simulation executes an "end simulation event" or the current simulation time exceeds the maximum time allowed for the simulation. The following algorithm depicts the actual execution:

```
schedule an initial event for each player
while ( current_sim_time < max_sim_time ) and ( event != end_event )
    begin loop
        wait until safe to execute
        get next event
        execute event
        schedule new event
    end loop
```

Players are allowed to move anywhere on the battlefield. This leads to players crossing sector boundaries and LP boundaries. Each player has a sensor with a specified range. Whenever a player can sense into a neighboring sector a copy of the player is sent to that sector. This player copy has a flag as one of its attributes that designates it as a copy. Player copies are used for two reasons:

1. It allows the neighboring sectors to predict events involving the player in question in its area.

2. It provides a smooth transition of players from sector to sector.

The transition of a player from Sector 1 to Sector 2 occurs as follows:

(a) The player is initially in Sector 1 and is headed into Sector 2.

(b) The player can sense into Sector 2; Sector 2 receives the player copy.

(c) The player's center of mass crosses into Sector 2. Sector 2's copy becomes the actual player. The set of attributes that were the player in Sector 1 become a player copy.

(d) The player copy in Sector 1 is deleted once the player can no longer sense into Sector 1.

Often sector crossings involve sectors on different LPs. When this occurs, the player copy is sent as a message to the destination LP.

3.3.4 Battlefield Events. Events drive this battlefield simulation model. Some examples of events are *startplayer*, *collisiondistancereached*, and *reachedturnpoint*. These events and others are explained in Appendix A.

3.4 Dynamic Load Balancing Model

The following sections describe issues that were considered in the design of the dynamic load balancing model for the general battlefield model.

THIS PAGE IS MISSING IN ORIGINAL DOCUMENT

23 3.4.1, 3.4.2

3.4.3 Task Migration Strategy. If it is determined profitable to load balance the system, then sources and destinations for task migration are determined. The sources are informed of the quantity and destination of tasks for load balancing.

Two approaches were analyzed for determining the source and destination processors:

1. Global - this approach allows any two processors in the system to become the source and destination processors. Although this approach provides great flexibility for load balancing, it would increase communication costs in a system with strong communication dependencies.
2. Local - this approach only allows two processors that are neighbors to become the source and destination processors. It limits the options for load balancing, but is useful in a system where there are strong communication dependencies.

3.4.4 Task Selection Strategy. The source processors, once informed of the quantity and destination of tasks, must determine the best tasks for efficient and effective load balancing and send them to the appropriate destinations.

The following approaches were considered:

- Moving players - transferring players between processors.
- Moving sectors - transferring one or more sectors between the source and destination processors
- Changing sector boundaries - Changing sector boundaries on the source processor to decrease sector size and transfer that extra battlefield space to the destination processor in the form of larger sectors.

3.5 Benchmark Scenarios

Once any load balancing model is designed and implemented, a method is needed to measure its performance. A straight forward approach would be to develop scenarios to compare the performance of the battlefield simulation with and without the load balancing model. These scenarios would then be known as *benchmark* scenarios and would play an important role in analyzing the load balancing model.

A benchmark is a point of reference from which measurements can be made, or the use of a program to evaluate the performance of a computer system. Gray proposes that a good benchmark is both relevant and scalable (6).

The hypothesis of this thesis states that dynamic readjustment of the spatial boundaries assigned to each LP should balance the work among LPs, and increase simulation speedup. Benchmark scenarios need to create the conditions necessary to prove or disprove this hypothesis. The condition that needs to be achieved is unbalanced work loads between LPs.

3.6 Conclusion

This chapter provided a discussion of the issues associated with the design of a dynamic load balancing model. Chapter 4 discusses the design of the model with the corresponding design decisions.

IV. Design and Implementation

4.1 Introduction

This chapter discusses the design and implementation of the load balancing model resulting from this research. The issues presented in Section 3.4 are analyzed and decisions are reached. These decisions and their rational are presented in this chapter.

4.2 Design

4.2.1 Processor Load Evaluation. The processor load value for a particular LP_i used by the general battlefield simulation model involves a four level hierarchy of calculations. The lowest level calculation is that of a player.

4.2.1.1 Player Load Evaluation. One barometer of the amount of work that the simulation performs is the number of events it has to process. Each event is associated with one or more players. One method to determine the load a certain player P_e has on the system would involve keeping track of the number of events executed for that player. The load for a particular player P_e is defined by the following formula:

$$load(P_e) = \frac{number\_of\_events\_for\_player_p \text{ in } time\_frame}{time\_frame}$$

The variable $time\_frame$ can extend from the beginning of the simulation until the current time or any subset of this time period. It uses a moving window average to allow the calculation for the load of a player to be more relevant to the current (and thus expected) load due to the player. For example, player A might have had numerous events early in

THIS PAGE IS MISSING IN ORIGINAL DOCUMENT

27

4.2.2
4.2.3

- Moving players - transferring players between processors would increase communication between processors considerably. The simulation would have to search for players on all processors when determining the next event for a particular player. The current implementation requires only a search of players in the same sector. All players in the same sector are on the same processor.
- Changing sector boundaries - this would involve a very inefficient process of searching for each player in the old and new sectors and determining if the player belonged in its original sector or the new enlarged sector.

The sector was chosen as the task to be transferred between processors for load balancing for the following reasons:

1. It was the most efficient to implement.
2. It maintains the integrity of the simulation.
3. It requires the least amount of updates to the simulation maps.

4.2.4 Task Migration Strategy. Two approaches were considered for task migration, global and local. The local approach was chosen. The global approach was not chosen because of communication costs. For example, refer to Figure 3. If LP 1 is allowed to load balance with any other LP on the battlefield, it can have communication dependencies with up to 3 other LPs. If it is allowed to load balance with only its physical neighbors on the battlefield, it will only ever have communication with two other LPs. In this example communication dependencies with the global approach is only three, but when the num-

ber of LPs grows to n, it would be n-1. This introduces a great deal of communication overhead.

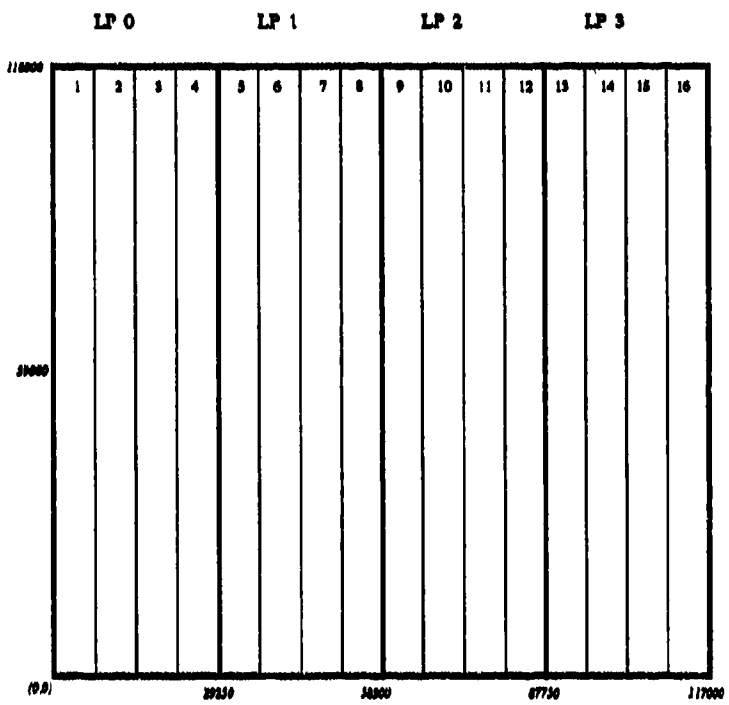


Figure 3. Battlefield Example

The local approach restricts an LP to load balance with only its left and right neighboring LPs on the battlefield. It keeps communication dependencies between LPs at a minimum. LPs are allowed to transfer sectors to and from their left and right neighbors in an effort to load balance. If an LP wants to send a sector to its neighbor, it has to send the sector physically located next to that neighbor. LPs must retain at least one sector at all times during the simulation.

4.2.5 Load Balancing Profitability Determination. An important portion of the design of this or any load balancing model is determining when it is profitable to load balance. This model makes that determination whenever an LP is considering sending one of its sectors to a neighboring LP in an effort to balance the work load. The cost of load balancing and the gain of load balancing is estimated to determine the profitability. Both of these estimates are functions of real time.

4.2.5.1 Cost of Load Balancing. The estimation of the cost of sending sector S_j from one LP to another is:

$$cost(S_j) = cost\_of\_packing(S_j) + cost\_of\_communication(S_j) + cost\_of\_unpacking(S_j)$$

The cost of packing a sector and the cost of unpacking the sector is based on the number of players in the particular sector. The cost of communication is also based on the number of players.

4.2.5.2 Gain of Load Balancing. The gain of sending sector S_j from one LP to another is the estimated reduction of simulation run time realized by the transfer. The estimation formula for the gain is:

$$gain(S_j) = time\_since\_last\_load\_bal\_event - time\_to\_next\_load\_bal\_event$$

The *time_since_last_load_bal_event* is the wall clock time since the last load balance event. The *time_to_next_load_bal_event* is a projection of the wall clock time to the next load

balance event if load balancing takes place. The *time_to_next_load_bal_event* is estimated as follows:

$$time\_since\_last\_load\_bal\_event * (\frac{load\_imbalance\_projection}{current\_load\_imbalance})$$

The *load_imbalance_projection* will be the load imbalance between LPs if the sector is transferred. The *current_load_imbalance* is the current load imbalance between the two LPs. The ratio of *load_imbalance_projection* to *current_load_imbalance* gives an estimate of how much the load imbalance will be improved by load balancing. If the ratio is less than one, load balancing will decrease the load imbalance; equal to one, stay the same; greater than one, increase the load imbalance. If the ratio is less than one load balancing will decrease the load imbalance and should decrease the wall clock time to the next event. This will give a positive gain which can then be compared to the cost.

4.2.5.3 Profit. The estimation of the profit of sending sector S_j from one LP to another is:

$$profit(S_j) = gain(S_j) - cost(S_j)$$

If this calculation is positive than it is profitable to load balance at this time, else it is not profitable.

4.2.6 Execution of Load Balancing Model. The execution of the load balancing model is driven by the load balance event. This event is scheduled at the beginning of the simulation on each LP. The *load_balance_period* specifies how often in simulation time this

event is to be executed. The following algorithm depicts the actual algorithm for the load balancing event:

```
synchronize with other LPs
calculate own load
determine possible destination
send load to possible destination
receive load from possible destination
if own_load > destination_load then
       while profitable to load balance
            send sector to destination
            update maps
else
       wait for possible new sector(s)
schedule next load balancing event(current_sim_time +
                                   load_balance_period)
continue simulation
```

Although the simulation time for each load balance event on different LPs is the same, some LPs might be running faster than others. The first step of the algorithm causes all LPs to be synchronized. This forces each LP of a pair to execute the load balance event at the same real time. This is important for the following reasons:

- Sending a sector between two LPs with different times could result in lost players. Players could be in the process of moving to the sector being moved. After the sector is moved, the players could arrive at the source LP and the sector would be gone.
- As a result of lost players, the load calculation, as well as the simulation, will be incorrect.

There is a performance cost associated with this synchronization. It is not as bad as it seems because the LPs are always in synchronization to a certain degree because of the conservative simulation protocol used. This concept is explained further in Chapter 6.

THIS PAGE IS MISSING IN ORIGINAL DOCUMENT

33 4.3
4.3.1

1. Host node performs the synchronization. In this case each LP would send a message to the host stating that it was ready to load balance. Once the host received this type of message with the same simulation time stamp from every LP, it would send a message back to every LP instructing it to go ahead and load balance.
2. LPs synchronize among themselves. Each LP would send a message to both of its neighbors stating it was ready to load balance. An LP could start load balancing when it got a message back from both of its neighbors stating it was okay to proceed.

The host synchronization option was utilized for this implementation for two reasons. First it requires less messages. Each LP will send and receive a message to/from the host. The second option would require each LP to send and receive a message from every other LP. Second it was very straightforward and already implemented in BATTLESIM for state saving.

4.3.2 Transferring a sector to another LP. Each LP knows about all sectors on the battlefield including their coordinates. Therefore when a sector is transferred, only its playerset and its associated player maps need to be sent. Performance on the iPSC/2 tends to favor longer messages than shorter ones so the playerset is sent in one long message (10). The playerset is queried for all players. Each player is packed in contiguous memory along with its associated maps. Once the playerset message is fully packed, it is sent to its destination LP.

4.4 *Conclusion*

It is interesting to note that the design of this load balancing model does not strictly follow any of the previously well known schemes. Once LPs pair off during the load balance cycle, the LP within each pair with the greatest load decides whether to load balance or not. In effect this LP becomes the sender, and the process continues similar to the sender initiated scheme discussed in the literature review. The design discussed in this chapter has taken ideas many ideas from different schemes and tailored them to the general battlefield model.

V. Testing, Results and Conclusion

5.1 Introduction

This final chapter discusses the functional testing of the load balancing implementation, results of performance testing, recommendations for further work, and conclusions. Functional testing was accomplished to validate the proper execution of the load balancing algorithm and to check that it maintained simulation integrity. Initial performance tests were performed and the results are presented and analyzed.

5.2 Functional Testing

The load balancing model was tested for functional considerations. The first test was to ensure that the LPs could synchronize correctly in response to a load balance event. This test worked for two and four LP configurations. The second test exercised the load calculation for an LP. A scenario containing sectors with zero, one and multiple players was used to verify this calculation. This test was successful. The third test verified the proper transfer of a sector from one LP to a different LP. A sector with several players was used to verify that the implementation properly handled this transfer. The fourth and final test consisted of a variety of scenarios to test the whole load balancing algorithm to make sure that the integrity of the simulation was not violated. Scenarios included sector crossings, LP crossings, and collisions for the LP and four LP configurations. A comparison of the output trace files between the load balancing runs and the non load balancing runs was used to ascertain consistency. All comparisons showed that the load balancing algorithm never violated the integrity of the simulation.

5.3 Performance Testing

5.3.1 Test Scenarios. Benchmark Scenario 1, depicted in Figure 4, was the base scenario used in the initial performance tests of the dynamic load balancing. Variations of this scenario were also used. The scenario was designed to run with two LPs. Each LP was assigned four sectors. This scenario was designed to demonstrate a load imbalance between LP 0 and LP 1. Each player in this scenario has 200 route points and is moving at a speed of 300 units per second. The dimensions of the battlefield were 118000 units by 117000 units. During the simulation each player moves forward and backward along its respective sectors. The performance of the load balancing algorithm was measured by comparing the execution time of BATTLESIM running with and without the load balancing algorithm for given scenarios and LP configurations.

5.3.2 Preliminary Results. Performance test 1 used Benchmark Scenario 1 and ran on two LPs with a minimum time delay of one. This scenario started with LP 0 having a load three times as great as LP 1. The load balancing simulation run detected the imbalance in the simulation and balanced the work load during the first load balance event. In Table 2 the first column represents the simulation with no load balancing, the second column with load balancing, and the third column represents the simulation with no load balancing but with a balanced work load from the start, and the fourth column the sequential simulation.

Performance test 2 involved a rerun of performance test 1 relaxing the minimum time delay. Table 3 shows the results.

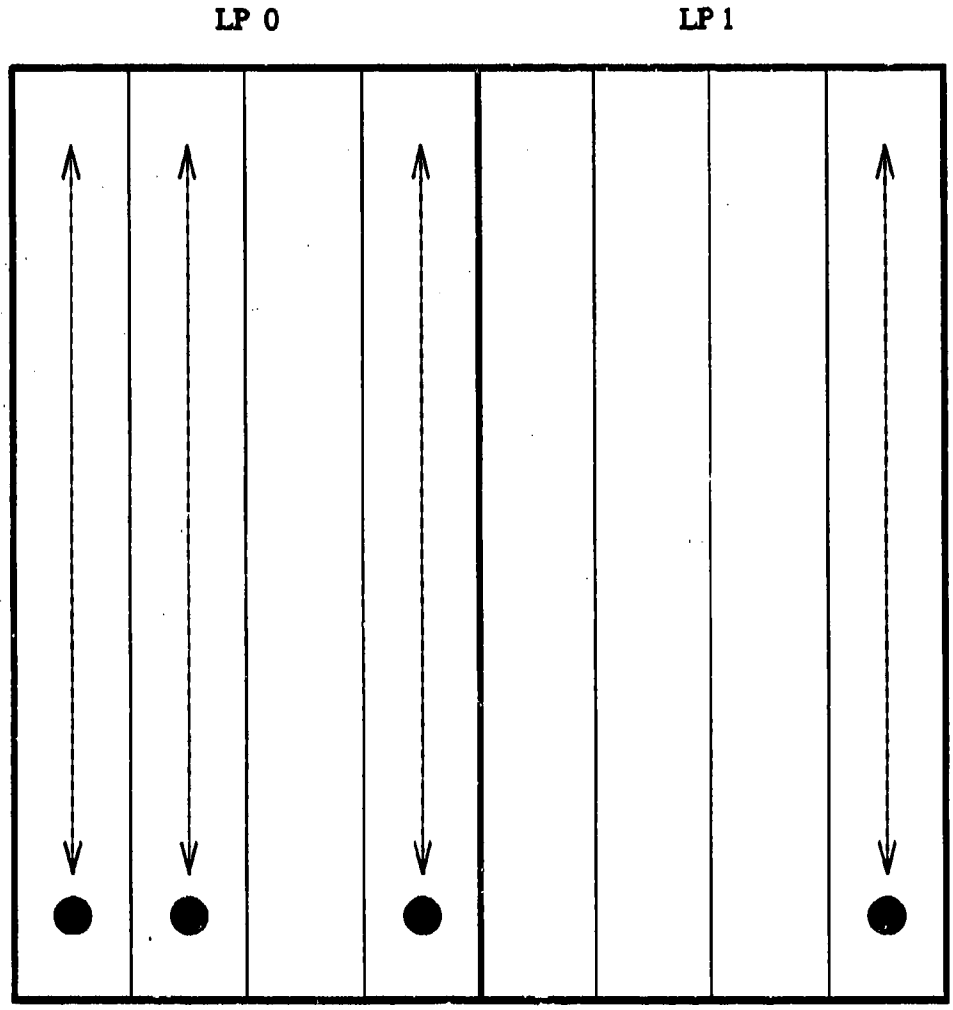


Figure 4. Benchmark Scenario 1

Table 2. Performance Test 1

Simulation Type	Run Time (secs)	Number of LPs
With load balancing	101.022	2
Without load balancing	106.013	2
Balanced	100.987	2
Sequential	18.339	1

Table 3. Performance Test 2

Simulation Type	Run Time (secs)	Number of LPs
With load balancing	13.487	2
Without load balancing	18.98	2
Balanced	13.131	2
Sequential	18.339	1

Performance test 3 involved a modification of Benchmark Scenario 1. The only parameter that was changed was the relative speed of the players. A minimum time delay of one was still used. Performance test 1, generated sets of events with the same simulation time. This was due to the players having the same speed and distance between route points. Table 4 contains the results of the run.

Table 4. Performance Test 3

Simulation Type	Run Time (secs)	Number of LPs
With load balancing	108.498	2
Without load balancing	108.797	2
Balanced	108.122	2
Sequential	18.345	1

5.3.3 Analysis. Performance test 1 showed approximately a five percent performance improvement between the non load balancing and load balancing runs. Performance test 2 showed approximately a five percent speedup between the sequential run and the load balancing run.

The run times of the first three simulation runs were very close in performance test 3. The surprising result is the difference between the balanced and non balanced non load balancing runs. This difference would expected to be greater. One possible explanation for this involves the synchronization protocol.

The LP synchronization protocol uses a minimum time delay, t_{min}, of one for all of its communication arcs. This has historically been one in BATTLESIM to prevent causality errors.

The top of Figure 5 depicts the physical structure of n LPs on the battlefield while the bottom of the figure depicts the communication graph for the LPs. Arcs into an LP specify all other LPs from which messages can be received. Arcs leaving an LP specify those LPs to which it can send messages. Each LP has two input arcs and two output arcs as a result of the stripwise partition of the battlefield. LPs 0 and n-1 have only one input arc and one output arc.

To prevent causality errors, an LP must wait to update its safetime until it receives a message on each input arc with a time at least equal to its own safetime. Once this constraint is satisfied, the LP identifies the smallest time stamp of these messages and updates its safetime to this time. The LP can then execute events in its next event queue up to this safetime.

The structure of the LP communication graph implies that LP_i must wait on LP_{i-1} and LP_{i+1} for messages in order to update its safetime. When an LP receives a message it sends a NULL message with a time stamp of min(t_NEQ, $safetime + mintimedelay$) on all of its output arcs. As a result, the difference in simulation times between any two

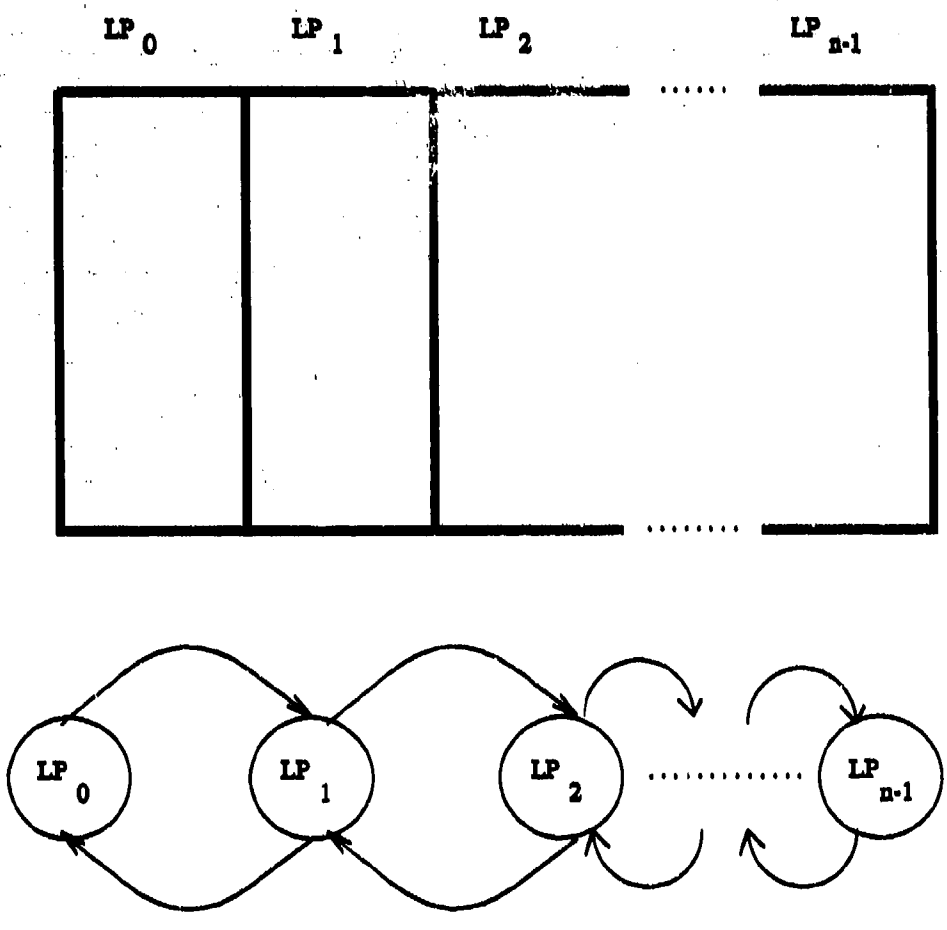


Figure 5. LP structure and communication graph

LPs will be less than or equal to the minimum time delay. This imposes a constraint of how much concurrent processing can occur during the simulation. This might explain why there was not a bigger difference in the run times between the load balancing and non load balancing runs. Testing was also accomplished with this in mind. Performance test 3 was rerun relaxing the minimum time delay. Results showed a speedup at the same rate for each type of simulation run. The difference in run times between the load balancing and non load balancing runs stayed the same.

A different explanation to problem might have been the fact that the events execute so fast, that a difference in number of events between LPs would generate a very small difference in the run times. A spin loop was implemented for each event type in an effort to prove or disprove this assertion. Results showed a performance degradation at the same rate for each type of simulation run. The difference in run times between the load balancing and non load balancing runs again stayed the same.

Another possible explanation to be explored might deal with the length of the NEQ. The simulation runs so far have only two events on the NEQ at any time instant. Even though there was a vast difference in the number of events processed by each LP during the different runs, the length of the NEQs stayed fairly constant. Scenarios should be developed to generate a load imbalance by varying the length of the NEQs between LPs. This could be done by generating scenarios with large number of players.

5.4 Recommendations for Further Research

There is still much work to be done in the topic of this thesis. The following list of items are recommendations for further research

- Generate larger scenarios than were used in this research. Larger scenarios with a larger number of players need to be created to determine if the load balancing algorithm designed in this thesis is effective or not.
- Investigate further the effect of changing the minimum time delay of the LP synchronization protocol to be dynamically calculated instead on being a constant. Test runs for scenarios in this research showed significant speedup improvement when the minimum time delay was increased.
- Modify the LP synchronization protocol to use float times for LP arc (channel) updates instead of integer times. Events occur at float times and synchronization may not be accurate when objects are migrating between LPs.
- Investigate further the effect of the load balancing algorithm on the LP synchronization protocol. A potential problem exists if an LP updates it safetime and then transfers one of its sectors as a result of load balancing. The LP could now generate a new safetime that could be less than its safetime before the sector transfer. This could result in a causality error.

5.5 Conclusion

A dynamic load balancing model was designed using the general load balancing model described by Willebeek-LeMair and Reeves (18). This model was implemented and the resulting algorithm was functionally tested. The algorithm effectively maintained the integrity of the system.

Initial performance tests showed a slight performance improvement between the non load balancing and the load balancing runs. Further tests with larger scenarios need to be run before a conclusion concerning performance can be drawn.

This research has demonstrated the feasibility of using dynamic load balancing as a performance improvement tool for parallel discrete-event battlefield simulations. The model developed will prove to be a valuable test bed for further research with the various load balancing parameters.

Appendix A. Battlefield Simulation Events

The following are events from the battlefield model:

- **Back End Object** – The Back End Object event indicates that the back end of an object has left a sector. The player will be removed from that sector and all appropriate object relationship maps will be modified to eliminate a reference to that sector as the location of a player copy.
- **Center of Object** – The Center of Object event indicates that the center of an object has moved to a new sector.
- **Collision Distance Reached** – This event indicates that a two player objects have collided. The event exchanges information between the players to indicate the speed and mass of the object they have collided with and the players respond accordingly.
- **Entered Sensor Range** – This event indicates that a player object has entered the sensor range or another player. The event notifies the player and the player responds accordingly.
- **Front End Object** – The Front End Object event indicates that the front end of an object has just entered a new sector. A copy of the player must be created in the new sector.
- **Made Sensor Contact** – The Made Sensor Contact event indicates that a player's sensors have made a contact. The event will indicate the contact to the player which will respond accordingly.

- **Reached Turnpoint** – The Reached Turnpoint event indicates that a player has reached one of its turnpoints.
- **Remove Player Copy** – The Remove Player Copy event indicates that a player copy is no longer in a given simulation sector and that it must be removed. This may occur as the result of a player leaving a sector or a player being destroyed.
- **Start Player** – The Start Player event indicates that a player needs to have its next event calculated. This Event is mainly used only at he beginning of the simulation to calculate the first event for each of the player objects.
- **Update Player Copy** –This event indicates that the player copy must be updated using the player copy attached to the event. If the player does not exist in this sector, then it must be created.

Bibliography

1. Bergman, Kenneth C. *Spatial Partitioning of a Battlefield Parallel Discrete Event Simulation*. MS thesis AFIT/GCS/ENG/92D-03, AD-A258911, Air Force Institute of Technology, 1992.
2. Chandy, K. M. and Jayadev Misra. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, *SE-5*(5):440–452 (Sep 1979).
3. Erciyes, K. and S. Yilmaz. "Dynamic Load Balancing in a Distributed Computer System," *IEEE Transactions on Parallel and Distributed Systems* (1993).
4. Fujimoto, Richard M. "Parallel Discrete-Event Simulation," *Communications of the ACM*, *33*(10):31–53 (Oct 1990).
5. Goswami, Kumar K., et al. "Prediction-Based Dynamic Load-Sharing Heuristics," *IEEE Transactions on Parallel and Distributed Systems*, *4*(6):638–648 (Jun 1993).
6. Gray, Jim. *The Benchmark Handbook for Database and Transaction Processing*. Morgan Kaufmann Publishers, 1991.
7. Hammond, Steven W. *Mapping Unstructured Grid Computations to Massively Parallel Computers*. PhD dissertation, RPI, 1992.
8. Hanxleden, Reinhard V. and L. Ridgway Scott. "Load Balancing on Message Passing Architectures," *Journal of Parallel and Distributed Computing*, *13*(3):312–24 (November 1991).
9. Kumar, Vipin, et al. "Scalable Load Balancing Techniques for Parallel Computers." Army Research Office grant 28408-MA-SDI to the University of Minnesota.
10. Lamont, Gary B. *Compendium of Parallel Programs for the Intel iPSC Computers*. Unpublished Report, Air Force Institute of Technology, Dec 1993.
11. Lewis, Ted G. and Hesham El-Rewini. *Introduction to Parallel Computing*. Prentice Hall, 1992.
12. Lin, Hwa-Chun, et al. *Performance Study of Dynamic Load Balancing Policies for Distributed Systems with Service Interruptions*. Technical Report, Department of Electrical Engineering-Systems, University of Southern California, 1991.
13. Misra, Jayadev. "Distributed Discrete-Event Simulation," *ACM Computing Surveys*, *18*:39–65 (March 1986).
14. Moser, Robert S. *A Spatially Partitioned Parallel Simulation of Colliding Objects*. MS thesis AFIT/GCS/ENG/91D-15, AD-A274217, Air Force Institute of Technology, 1991.
15. Office of Deputy Secretary of Defense. *Modeling and Simulation Management Plan*, June 21 1991. Version 1.1.
16. Trachsel, Walter G. *Object Interaction in a Parallel Object-Oriented Disrete-Event Simulation*. MS thesis AFIT/GCS/ENG/93D-03, AD-A274084, Air Force Institute of Technology, 1993.

17. Van Horn, Prescott J. *Development of a Protocol Usage Guideline for Conservative Parallel Simulations.*. MS thesis AFIT/GCS/ENG/92D-19, AD-A258851, Air Force Institute of Technology, 1992.

18. Willebeek-LeMair, Marc H. "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993 (Sep 1993).

Vita

Captain Seth R. Guanu was born November 25, 1967, in Watertown, New York. After graduating from Cherry Valley Central School in 1985, he enrolled in the State University of New York (SUNY) at Potsdam and graduated with a Bachelor of Arts in Computer Science and Mathematics. While at SUNY, he earned a Reserve Officer Training Corps scholarship and was commissioned as a second lieutenant in May of 1989.

Captain Guanu's first assignment was with the 50th Space Systems Squadron at Falcon AFB as the Chief of the Computer Security Division. He was responsible for all facets of computer security at the Consolidated Space Operations Center (CSOC). Captain Guanu's subsequent position with the 50th was as Chief of the Computer Performance Section.

He entered AFIT in May of 1992.

Permanent address: 22 Montgomery Street
Cherry Valley, NY 13320

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1994	Master's Thesis

4. TITLE AND SUBTITLE
Dynamic Load Balancing for a
Parallel Discrete-Event Battlefield Simulation

5. FUNDING NUMBERS

6. AUTHOR(S)
Seth R. Guanu, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94J-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt Rick Painter
2241 Avionics Circle, Suite 16
WL/AAWA-1 BLD 620
Wright-Patterson AFB, OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis investigates issues involved in developing a dynamic load balancing model for a parallel discrete-event battlefield simulation. The research covers issues in task management, discrete-event simulation, parallel simulation, and load balancing. There are four primary issues discussed concerning the design of a dynamic load balancing model. The first issue is processor load evaluation which deals with the calculation of the amount of work on a processor. The second issue is load balancing profitability determination which deals with the decision to load balance or not based on some cost-gain relationship. The third issue is task migration which deals with the selection of sources and destinations for task migration. The fourth issue deals with task selection which involves selection of appropriate tasks for efficient and effective load balancing. As a result of the research, a dynamic load balancing model is designed that balances the work load in a parallel discrete-event battlefield simulation. The design goals used to develop this model were efficiency and maintainability of simulation integrity. The model is then implemented and tested using AFIT's BATTLESIM program, which is a battlefield parallel discrete-event simulation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Task management, Discrete-Event Simulation, Parallel Simulation, and Load Balancing	49
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL